Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Центр післядипломної освіти

Кафедра _____Програмної інженерії_____

(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти – другий (магістерський)

Дослідження та оптимізація алгоритмів створення, рендерінгу та обміну синтетичними даними з системами машинного навчання

(тема)

Виконала: студентка 2 курсу, групи ІПЗмзд – 18 – 1_____

_____Тихонова О. О._____

(прізвище, ініціали)

спеціальності 121 – Інженерія програмного забезпечення____

(код і повна назва спеціальності)

____Освітньо–наукової програми_____

(тип програми)

____Інженерія програмного забезпечення_____

(повна назва освітньої програми)

Керівник _____к. т. н., доц. каф. ПІ Вечур О. В._____

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З. В. Дудар

20___ р.

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Центр післядипломної освіти

Кафедра _Програмної інженерії_____

Рівень вищої освіти – другий (магістерський)

Спеціальність _121 – Інженерія програмного забезпечення_____

(код і повна назва)

Тип програми _освітньо–наукова програма_____

Освітня програма _Інженерія програмного забезпечення_____

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«___» _____ 20___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові _____Тихоновій Олені Олександрівни_____

(прізвище, ім’я, по батькові)

1. Тема роботи _Дослідження та оптимізація алгоритмів створення, рендерінгу та обміну синтетичними даними з системами машинного навчання_____

затверджена наказом університету від «27» березня 2020 р. № 41Стз

2. Термін подання студентом роботи до екзаменаційної комісії 26 червня 2020 р._

3. Вихідні дані до роботи *алгоритми обробки графічних даних, алгоритми створення синтетичних відео та зображень, систематизовані фактори впливу на результати машинного навчання. Використовувати ОС Windows, MacOS середовище Unity, Unreal Engine*_____

4. Перелік питань, що потрібно опрацювати в роботі *мета дослідження, аналіз практичного застосування результатів, огляд методів створення зображень та відео з сцен ігрових движків розгляд впливу текстур, ефектів після обробки, послідовності рендерінгу, вибір використаних ігрових движків*_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз предметної галузі	05.04.2020	Виконано
2.	Підготовка пояснювальної записки	01.05.2020	Виконано
3.	Підготовка презентації та доповіді	09.05.2020	Виконано
4.	Нормоконтроль, рецензування	16.05.2020	Виконано
5.	Попередній захист	22.05.2020	Виконано
6.	Занесення диплома в електронний архів	22.05.2020	Виконано
7.	Допуск до захисту у зав. кафедри	22.05.2020	Виконано

Дата видачі завдання «___» _____ 20___ р.

Студентка _____

(підпис)

Керівник роботи _____ к. т. н., доц. каф. ПІ Вечур О. В.__

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить**:** 69 с., 19 рис., 3 дод., 12 джерел.

СИНТЕТИЧНІ ДАНІ, PBR, BAKED, REALTIME, РЕНДЕРІНГ, ІГРОВИЙ ДВИЖОК, UNITY, UNREAL ENGINE, ТЕКСТУРА, КАРТА, ПАЙПЛАЙН.

Метою роботи є дослідження алгоритмів та інструментів створення, рендерінгу та обміну синтетичними даними з системами машинного навчання.

Методи дослідження базуються на застосуванні різноманітних підходів та технік до візуалізації зображення, використанні сучасних інструментів для створення фізично–реалістичних симуляцій, наприклад, – ігрових движків.

В результаті роботи було виведено ряд критеріїв за якими генерацію синтетичних даних та обмін ними з системою навчання можна пришвидшити та покращити результати її прогнозів, а також було створено програмне рішення, яке дозволяє експериментально підтвердити об’єкт дослідження на базі ігрового движку Unity.

SYNTHETIC DATA, PBR, BAKED, REALTIME, RENDERING, GAME ENGINE, UNITY, UNREAL ENGINE, TEXTURE, MAP, PIPLINE.

The purpose of this work is to study algorithms and tools for creating, rendering and exchanging synthetic data with machine learning systems.

Research methods are based on the application of various approaches and techniques to image rendering, the use of modern tools to create physically–realistic simulations, for example, – game engines.

As a result, a number of criteria were derived by which the generation and synthesis of synthetic data with the training system could accelerate and improve the results of its predictions, and a software solution was created to experimentally validate a research object based on the Unity game engine.

ЗМІСТ

ВСТУП

З кожним днем сучасний світ змінюється та перетворюється. Технології «еволюціонують» та становляться все більш адаптовані під потреби конкретного користувача. Головним інструментом створення індивідуалізованого програмного і апаратного забезпечення стають системи машинного навчання та штучного інтелекту. Однак, щоб ефективно ними користуватися – потрібні дані, які б могли бути застосовані для вирішення поставленої задачі. Трапляється так, що виникають такі задачі, де дані для навчання систем, які зможуть їх вирішити, – або взагалі відсутні, або їх отримання може коштувати значних ресурсів. Прикладом однієї з таких задач може бути створення системи, яка буде аналізувати стан помешкання та визначати чи є загроза пожежі, чи ні. Для цього нам потрібні відео, власне з пожежами, але здобути їх у достатній кількості не є можливим, а відтворювати власноруч – дорого та непрактично. Механізмом, який допоможе розробнику подібних систем, стають інструменти генерації синтетичних даних. Синтетичні дані – це штучно створена інформація за допомогою певних алгоритмів та лімітів, а не з реальних подій. У рамках цієї роботи, буде розглянуто проблеми, інструменти та способи створення саме візуальних (зображення, відео) синтетичних даних. Якщо переглянути всі існуючі програмні продукти у домені графічних редакторів, систем рендерінгу, пакетів 3D моделювання та ігрових движків, найбільш яскравими виступають саме останні. Причин цьому велика кількість:

- можливість процедурно створювати сцени будь–якої складності;
- використання декількох камер та сесій одночасно;
- можливості зручно та динамічно налаштовувати параметри рендерінгу, світла та камери;
- широкий спектр оптимізаційних рішень для пошвидшення роботи;
- реалістична поведінка світла у сцені (PBR);

– ефекти пост–обробки для камери;

– створення ефектів з використанням GPU.

Тобто сучасні ігрові движки, які використовуються для створення фізично–реалістичних симуляцій у кіно та при створенні ігор – також ідеальні для створення псевдовипадкових сцен, бо мають можливості гнучко налаштовувати різноманітні параметри. Це дозволяє кожен раз генерувати нову ситуацію, що дає можливість наближатися до реалістичних результатів машинного навчання та покращувати точність прогнозів. Беручи до уваги актуальність цієї проблематики та релевантність ігрових движків та систем рендерінгу у її вирішенні, метою даного дослідження є пошук кореляцій різноманітних параметрів налаштування сцен та їх рендерінгу у ігровому движку при генерації синтетичних даних, а також рішень на етапі моделювання та текстурування з точністю прогнозів системи машинного навчання. Другою задачею є порівняння різних движків у цьому аспекті, а також оптимізація процесу генерації даних під певну ситуацію. Для безпосередньої перевірки результатів, є релевантним використанням популярних та перевірених часом систем та алгоритмічних баз, наприклад таких, як TensorFlow, які використовуються у великій кількості в сучасних проектах. Якщо результати дослідження покажуть достатню ефективність системи, то це доведе можливість створення більш точних моделей у випадку більш спеціалізованих програмних пакетів для машинного навчання [1].

Слід зазначити, що елементами наукової новизни у цьому питанні є систематизовані та зважені аспекти, які впливають на результати машинного навчання при створенні синтетичних даних, а також створення технік оптимізацій цього процесу. Практичним значенням цього дослідження є можливість впровадити системи штучного інтелекту та машинного навчання у такі домени життя, в яких наразі це неможливо. А також, застосування у наукових дослідженнях в певних галузях, покращення тестування вже існуючих програмних забезпечень та продуктів, яке дозволить розширити можливості розробників у створенні власних прототипів у обраній галузі з реального життя.

1 ОПИС ПРОВЕДЕНИХ ТЕОРЕТИЧНИХ ДОСЛІДЖЕНЬ

Основна проблема синтетичних наборів даних – це наблизити їх до подібності з реальним випадком використання, особливо відео. Але справжні проблеми виникають, коли прогнози зосереджуються на дрібних деталях, таких як розпізнавання обличчя. Крім того, рендерінг довгих фотореалістичних відео або зображень може бути надзвичайно важкою операцією. Але це можна подолати, наприклад, згенерувавши карти сегментації. Моделі, які можуть бути навчені синтетичними даними, обмежені насиченістю та «багатством» даних, які ми можемо отримати. Що стосується зображень, які створені за допомогою програмного забезпечення для 3D рендерінгу, то є можливість повторення значної частини реального зображення завдяки науковому розумінню світла та фізики.

Розглянемо також деякі інші способи генерації візуальних синтетичних даних, окрім використання ігрових движків, щоб мати більш повну картину переваг саме цього методу. Вирізання та вставка одного зображення в інше виглядає як перспективна ідея в цьому плані. У цьому аспекті є можливість вирізати набір предметів та підставляти їх до різноманітних оточень.

Проблема цього механізму полягає в тому, що це надзвичайно важко вирізати та вставити предмет достатньо природнім та реалістичним виглядом; тому це потребує додаткових модифікації зображення методами аугментації та вдосконалення. Це н надто важливо розміщувати об’єкти природньо в глобальному масштабі зображення, але важливо досягти локального реалізму. Сучасні класифікатори об’єктів не так критично, щоб мати об’єкт, наприклад на столі чи на підлозі; однак важливо змішати об’єкт якомога реалістичніше на локальному тлі. Навіть досить наївна вставка об’єктів може допомогти покращити виявлення об’єктів у системах машинного навчання, зробивши по суті синтетичні дані. Наступним кроком у цьому напрямку було б створити вставлені об’єкти бути узгодженими з геометрією та іншими властивостями сцени. У цьому контексті є такі окремі випадки, як наприклад – локалізація тексту, тобто виявлення об’єктів

спеціально для тексту, що з’являється на зображенні. Однак, усі ці методи виглядають не достатньо ефективними та швидкими, адже потребують дуже специфічних підходів до їх імплементації. Саме тому будемо розглядати саме ігрові движки у якості основного інструмента.

Отже, першочерговою задачею є вибір ігрового движка для створення тестових даних. Буде розглянуто лише open–source чи умовно безкоштовні рішення, бо доступ та ліцензії до комерційних та внутрішніх движків різних компаній відсутні. Движки, які працюють лише з 2D графікою (cocos2dx, godot та інші) у дослідженні розглядаться не будуть, бо поставлена задача полягає у розгляненні саме залежності фотореалістичних ефектів до результатів навчання моделей.

Основними рішеннями на ринку рішень для створення ігор є Unreal Engine 4 (далі – UE4, Unreal) та Unity. Ці движки є основними конкурентами та знаходяться у перегонах, хто з них може називатися найбільш developer–friendly. Вони дуже схожі за набором можливостей, але реалізація та пріоритети у них досить різні. Unity є рішенням більш загального призначення з точки зору крос–платформності, що не є вирішальним фактором у роглянутому випадку. UE4 –сфокусований на пристрої високого класу (PC, консолі PS та XBOX). Також, Unreal зарекомендував себе у створенні VR програм та візуальних ефектів у кіноіндустрії. Однак, однією з метрик за якою будуть оцінюватись результати дослідження є час рендерінгу та обробки зображення, тому Unity може бути швидшим у створенні даних. У цьому випадку UE4 може стати не гнучким та нерелевантним.

Отже, виділяємо метрики за якими будемо оцінювати навчання моделей з синтетични даними:

- точність прогнозу;
- девальвація від навчання моделей за реальними даними;
- швидкість створення даних;
- швидкість рендерінгу;
- розмір створених файлів;

– розмір одного «instance» для роботи системи (розмір вихідного коду движків може займати велику кількість місця).

Наступним етапом є виділення параметрів, які можуть впливати на результати у рамках доступних функцій движка, таких як:

– кількість елементів у сцені;
– формати використаних моделей (OBJ, FBX тощо);
– кількість полігонів у моделях;
– деталізованість текстур;
– UV розгортка;
– розширення текстур;
– формати використаних текстур (PNG, TGA, JPEG);
– використання відео чи зображень;
– адаптивність під різні типи задач;
– використання PBR текстур;
– використовування hand–paint текстур;
– налаштування параметрів світла – Baked vs Real time;
– кількість джерел світла;
– налаштування постпроцесингу камери;
– налаштування позиції камери (зум, кути огляду, ширина об’єктиву);
– використання ефектів часток (підрахування на CPU / GPU);
– використовування різних пайплайнів рендерінгу (у Unity, наприклад, HDRP – High Definition Render Pipeline, LWRP – Low Weight Render Pipeline, UWRP – Universal Render Pipeline);
– параметри налаштування матеріалів.

З точки зору системи навчання моделі для класифікації зображення, розглянемо такі можливості та параметри:

– швидкість навчання;
– стиснення зображень;
– точність прогнозів (у порівнянні з іншими системами);

– можливість використовувати згорткові нейронні мережі;

– можливість сегментації зображень;

– можливість візуалізувати результати.

Розглянемо кожен з пунктів та методи маніпуляції ними під час генерації даних.

1.1 Розгляд впливу текстур

Динамічна зміна кількості полігонів можлива лише за наявністю декількох однакових моделей з різними рівнями деталізації. У загальному сенсі кількість полігонів не є важливим фактором з точки зору класифікації зображення, бо навіть для низькополігональних моделей основна деталізація доводиться саме на текстури. Тому цей показник можна вважати вторинним.

Деталізованість текстур визначається наближенням до вигляду з реального світу. Це досягається за допомогою використання PBR (Physically Based Rendering) – набору технік візуалізації, в основі яких лежить теорія, наближена до реальної теорії поширення світла.

Для того, щоб рендер движка міг інтерпретувати текстуру,як фізично коректну, потрібно створити не тільки карту кольору (альбедо), але ще й карту нормалей, карту жорсткості, карту висоти, карту металевості та карту оклюзії.

Аналіз та рендерінг з використанням цих карт займає набагато більше часу та пам’яті, ніж з використанням hand–paint, де світло та тіні вмальовуються на пряму в карту кольору. Але, якщо є потреба в саме реалістичному зображенні, то іншого способу, окрім використання PBR, у даний момент немає. Тому, слід проаналізувати етапи та структуру фізично коректного рендерінгу та знайти можливі оптимізації які можуть знадобитися у ході дослідження.

Текстура (карта) альбедо визначає для кожного пікселя колір поверхні або базову відбивну здатність, якщо цей піксель металевий [2].

Карта нормалей дозволяє поставити для кожного фрагмента поверхні свою нормаль, щоб створити ілюзію, що ця поверхня більш випукла. Зазвичай усі інші карти є похідними від карти нормалей та можуть бути згенеровані за її допомоги, бо саме вона надає основну візуальну форму та деталізацію.

Карта жорсткості вказує, наскільки жорстка поверхня знаходиться в основі пікселя текстури. Обране значення з текстури жорсткості визначає статистичну орієнтацію мікрограней поверхні. На жорсткішій поверхні виходять більш широкі і розмиті відображення, тоді як на гладкій поверхні вони сфокусовані та чіткі. Карта жорсткості інтерпретується у якості хаотичного розподілу мікрограней серед яких частина світла поглинається. Можна сказати, що ці мікрограні представлені у якості маленьких зеркал серед яких розсіюється вхідний потік світла. Результатом такого розміщення є те, що промені світла розсіюються в різних напрямках на жорсткіших поверхнях, що призводить до більш яскравого блиску. І навпаки на гладких поверхнях променій з більшою вірогідністю відіб’ються в одному або паралельних напрямках, що дасть менш різкий відблиск.

Карта металевості визначає, чи є піксель металевим, чи ні. Залежно від того, як налаштований PBR движок, можна задавати металічність, як відтінками сірого, так і тільки двома кольорами: чорним та білим. Зазвичай ця карта визначає, то наскільки сильно об’єкт може відзеркалювати світло. Чим темніший піксель тим більше світла він відбиває, надаючи поверхні блиску.

Карта висоти (також відома як відображення паралакса) є концепцією, аналогічною відображенню нормалей, однак цей метод є більш складним – та, отже, більш дорогим. Такі карти зазвичай використовуються разом з картами нормалей, та часто вони застосовуються для надання додаткової чіткості поверхням.

Карти фонового затінення або AO визначають додатковий коефіцієнт затінення поверхні та навколишньої геометрії. Якщо, наприклад, у нас є цегляна поверхня, то текстура альбедо не повинна мати ніякої інформації про затіненні

всередині щілин цегли. Карта AO якраз визначає ці межі затемнення, за які важче проникнути світлу. З огляду на цю карту в кінці етапу освітлення, є можливість значно підвищити візуальну якість сцени. Текстури AO для геометрії та поверхонь або генеруються вручну, або попередньо обчислюються в програмах 3D моделювання. Саме ця текстура вносить найменше деталізації, але повноцінно займає час для рендерінга.

Порівняємо зображення сцени з наявністю AO текстури та без. Нижче на рисунку 1.1 зображено сцену, в якій використовуються карти AO для присутньої геометрії.

Рисунок 1.1 – Сцена з використання AO текстур

На результуючому зображенні карта AO надає незначних змін до освітлень внутрішніх деталізацій об’єктів. Не дивлячись на те що наша сцена майже повністю побудована з гладких об’єктів ми не побачимо майже жодної різниці. Однак, якщо наша сцена включає в себе велику кількість ерозійних об’єктів, такик як камені, скелі, дерева, ґрунт тощо, то важливість використання для надання більш реалістичного освітлення значно зростає.

На рисунку 1.2 зображено ту ж саму сцену, але в якій не використовуються карти AO для присутньої геометрії.

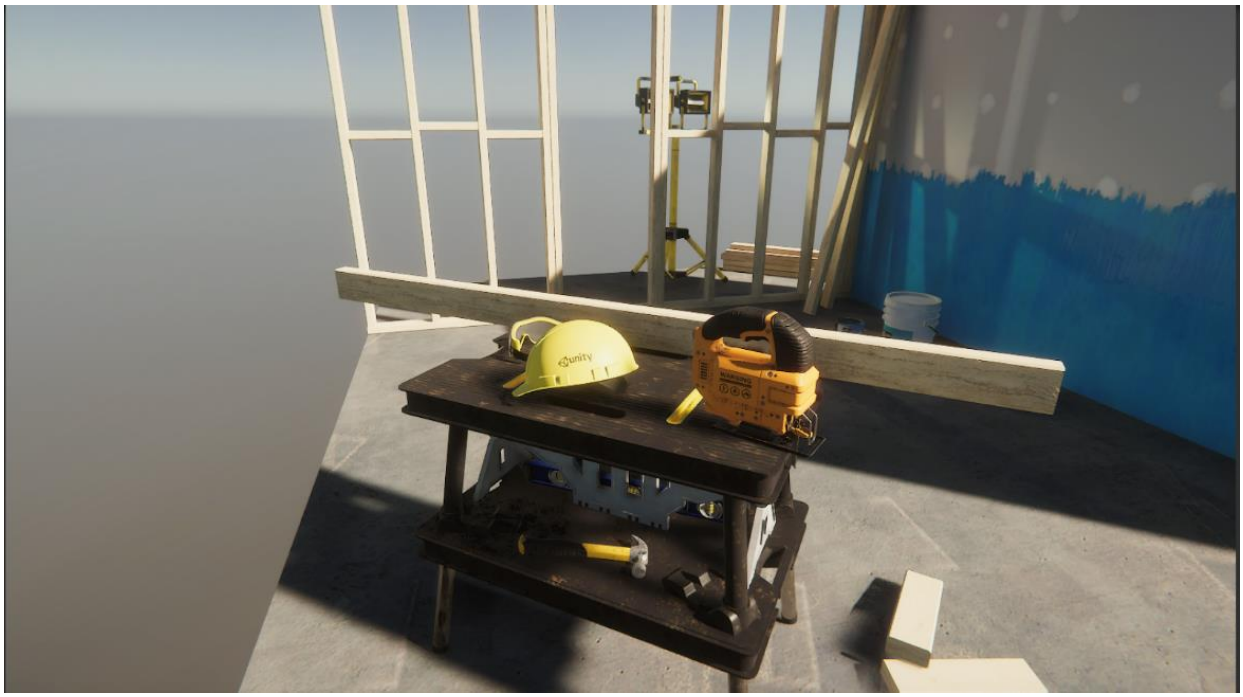

Рисунок 1.2 – Сцена без використання AO текстур

Неозброєним оком можна побачити, що різниця не є суттєвою, тому є доцільним не використовувати її під час генерації даних, звісно, якщо це не є винятком. Таким чином, підбиваючи підсумки аналізу впливу різноманітних текстур у фізично–коректному рендерінгу на результуюче зображення, можна стверджувати, що він повністю залежить від контексту задачі. Але у загальному випадку – є можливість нехтувати картою AO.

1.2 Порівняння пайплайнів рендерінгу

У рамках Unity одним з найвагоміших аспектів є обрання пайплайну рендерінгу зображення – UWRP чи HDRP.

HDRP – це високоточний пайплайн рендерінгу для сучасних (сумісних з Compute Shader) платформ. HDRP використовує різноманітні методи освітлення, які засновані на фізичних законах, елементах лінійного освітлення та освітлення HDR, та який використовує гібридну Tile / Cluster, Deferred / Forward архітектуру побудови світла. Він надає інструменти, необхідні для створення додатків, таких як ігри, технічні демонстрації та анімації з високим графічним стандартом. Тобто, HDRP – це рішення для покращення графічного вигляду за рахунок великої продуктивності сучасних комп'ютерів, а саме тому рендерінг та пост–обробка зображення займає багато часу [3].

Нижче на рисунку 1.3 можна побачити сцену з використанням пайплайну UWRP.

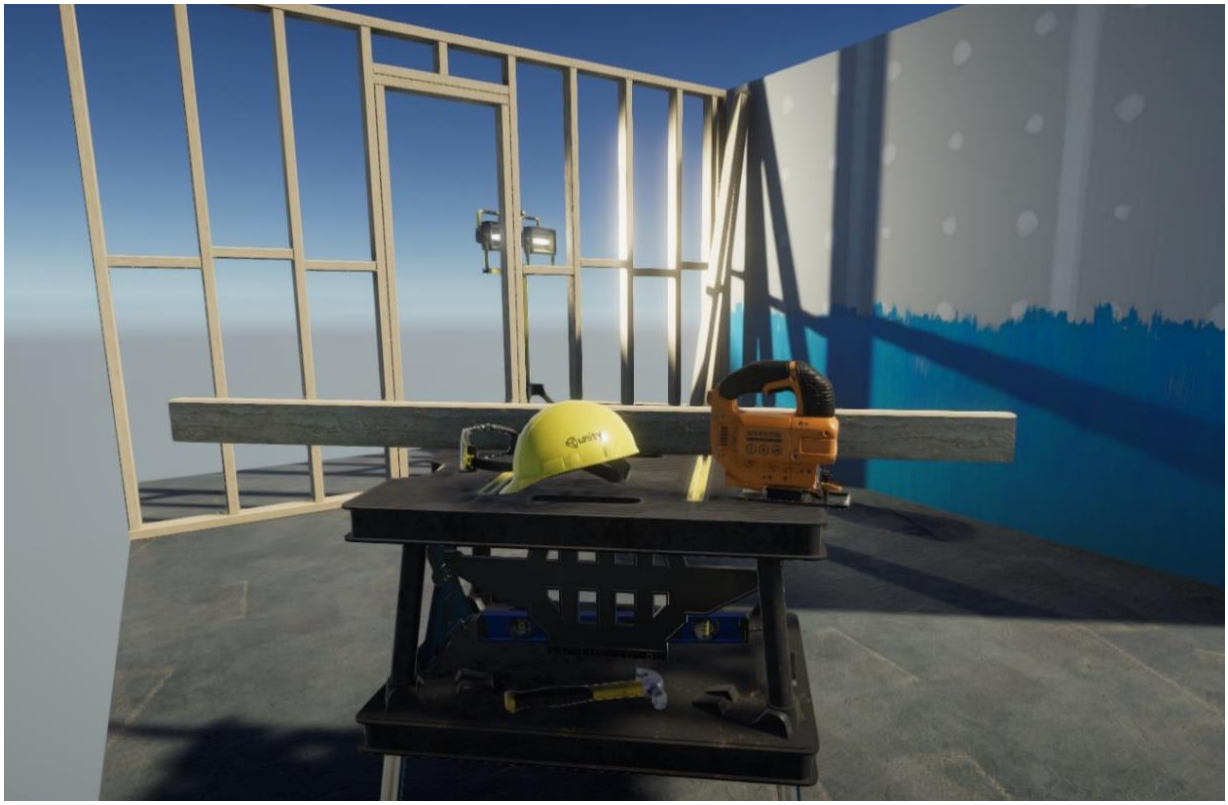

Рисунок 1.3. – Сцена з використанням пайплайну UWRP

UWRP – це більш легкий пайплайн з деякими графічними компромісами для покращення та пришвидшення рендерінгу. Мета UWRP – бути оптимізованим та

ефективним у реальному часі на платформах з обмеженою продуктивністю, здійснюючи деякі компроміси щодо освітлення та затінення. Цільова аудиторія для UWRP – це розробники, які орієнтовані на широкий спектр мобільних платформ, VR та тих, які розробляють ігри з обмеженими потребами в освітленні в режимі реального часу.

UWRP здійснює одноступеневу візуалізацію з одним тіньовим світлом у реальному часі з перевагою, що всі джерела світла затінені за один виклик відмалювання.

У порівнянні зі застарілим рендерінгом, який виконує додатковий виклик на кожен освітлений піксель в межах діапазону, використання цього пайплайну призведе до меншої кількості викликів відмальовування зображення. Це відбувається за рахунок деякої додаткової ускладненості шейдера за рахунок збільшення кількості джерел на один прохід функції відмальовування.

Порівнюючи два зображення однієї сцени, є можливість побачити різницю в освітленні та в її загальному вигляді, але основні елементи все ще можна добре розпізнати та основна деталізація не була загублена. Тому можна зробити висновок, що для загального випадку найкращим рішенням у навчанні моделі буде використання саме UWRP.

Також існує два методи зображення світла та тіней – baked та real–time lightning. «Запечене» світло – це світло, яке буде підраховано для кожного об’єкту один раз та залишиться незмінним. Ця техніка найбільш ефективна для статичних об’єктів. Використання зображень або статичних відео – це безсумнівно найкращий вибір у розглянутому випадку.

Але, якщо ми захочемо навчати нашу систему за допомогою відео (а саме у випадку, коли модель має відрізняти деталі з потоку зображень), слід використовувати підрахунок світла в реальному часі. Даний метод є менш продуктивним, але додає реалізм та динаміку. Ця ситуація дуже схожа з порівнянням UWRP / HDRP – балансування реалістичності та кращої графіки зі швидкістю та оптимізованістю [4].

Розглянемо безпосередньо значення при рендерінгу з використанням двох різних пайплайнів, але однієї й тієї ж сцени. Нижче на рисунку 1.4 зображено результати профайлінгу для UWRP.

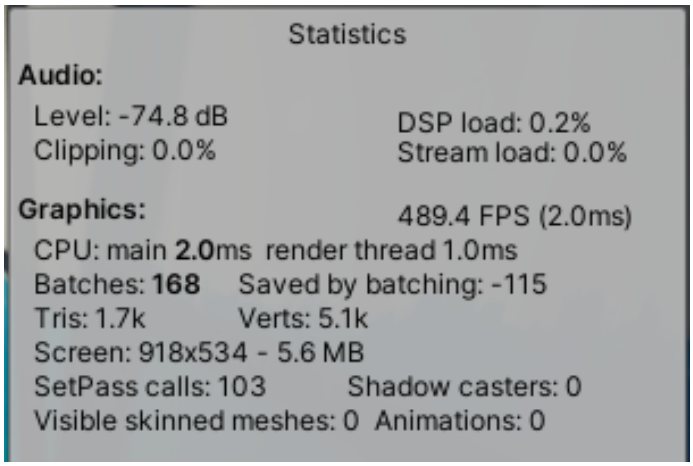

Рисунок 1.4 – Результати профайлінгу для UWRP

Та на рисунку 1.5 можна побачити результати профайлінгу для High Definition Render Pipeline.

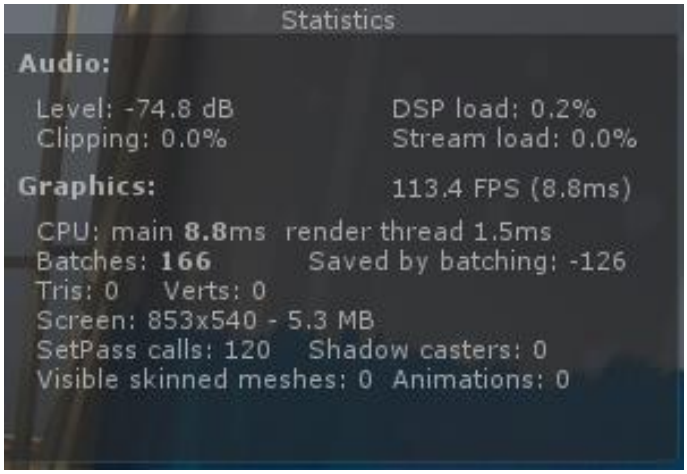

Рисунок 1.5 – Результати профайлінгу для HDRP

Результат профайлінгу показує, що повний процесорний час збільшився майже у 4 рази, а час безпосередньо для потоку рендерінгу – на 50%.

З точки зору оперативної пам’яті суттєвої різниці немає. Основна розбіжність – у кількості промальованих кадрів за секунду, майже в 4 рази. Це означає, що є можливість зробити висновок, що дійсно, використання UWRP допоможе сгенерувати більше даних за одну одиницю часу. В тому числі це означає, що ми можемо швидше отримати результат у вигляді навченої моделі. Далі при розгляданні цього питання слід оперувати результатами підрахунків точності прогнозу.

1.3 Вплив ефектів пост–обробки

Порівняємо хід роботи рендерінгу у UE4 та Unity. Система рендерінгу в Unreal Engine 4 базується на DirectX 11, включає в себе раніше згаданий Deferred / Forward освітлення, глобальне освітлення, просвічуваність і пост–обробку, а також системи частинок створених за допомогою GPU, використання векторних полей, тощо.

У випадку Unreal, немає розгалуження на різні типи інтерпретації сцени, але присутнє дуже тонке її налаштування. В загальному сенсі графічно Unreal Engine виглядає кращим та реалістичнішим за Unity, але як можна побачити на наведеній статистиці кадрів – він значно повільніший, якщо порівнювати з UWRP та приблизно однаковий з HDRP [5].

Unreal Engine 4 надає кілька ефектів пост–обробки, що дозволяє налаштувати загальний вигляд сцени. Приклади елементів та ефектів включають в себе – bloom (ефект HDR на яскравих об’єктах), оклюзію навколишнього середовища та відображення тонів.

Bloom – це світлові явища реального світу, які можуть значно доповнити реалізм зображення за помірні витрати на ефективність візуалізації.

Цей ефект можна побачити неозброєним оком, дивлячись на дуже яскраві предмети, що знаходяться на значно темнішому тлі. Слід зазначити, що подібні

ефекти пост–обробки камери для Unity доступні тільки для HDRP, тому неможливо їх використовувати у повному об’ємі та з такими ж параметрами оптимізації, що є у Unreal Engine 4.

Нижче на рисунку 1.5 можна побачити сцену з використанням ефекту Bloom.

Рисунок 1.5 – Сцена з ефектом Bloom

Bloom – це світлові явища реального світу, які можуть значно доповнити реалізм зображення за помірні витрати на ефективність візуалізації.

Цей ефект можна побачити неозброєним оком, дивлячись на дуже яскраві предмети, що знаходяться на значно темнішому тлі. Слід зазначити, що подібні ефекти пост–обробки камери для Unity доступні тільки для HDRP, тому неможливо їх використовувати у повному об’ємі та з такими ж параметрами оптимізації, що є у Unreal Engine 4.

Велику роль у створенні сцени можуть також відігравати візуальні ефекти, наприклад, відображення пожежі, вогню, диму тощо. У розглянутих движках є різні методи та способи створення візуальних ефектів, розглянемо їх безпосередньо.

У Unity є два способи ефективно створювати візуальні ефекти – стандартна Particle system та VFX Graph. В загальному сенсі система частинок (Particle System) імітує та візуалізує багато маленьких зображень чи 3D моделей, названих частинками, для створення візуального ефекту. Кожна частинка в системі являє собою окремий графічний елемент у ефекті. Система моделює спільно кожну частинку, щоб створити враження цільного ефекту. Системи частинок корисні, коли ми хочемо створювати такі динамічні об’єкти, як вогонь, дим або рідини, тому що важко зобразити такий тип об’єкта за допомогою геометрії у 3D або спрайтів (зображень) у 2D. Моделі та спрайти (sprites) краще підходять саме для зображення твердих предметів, такі як будинок чи автомобіль. Стандартна система частинок надає можливість редагувати її параметри з коду програми, яка виконується.

Ми можемо використовувати API системи частинок для створення власної поведінки, тобто не потрібно кожен раз створювати нову систему частинок для того, щоб зробити деякі зміни в ефекті, достатньо лише змінити доступні параметри налаштувань. Це безумновно надає гнучкості системі, бо є можливість реалізувати такі механізми, як поступове затухання вогню тощо. У класичної системи часток є можливість змінювати такі параметри як гравітаційний вплив навколишніх об’єктів на частинки, початковий розмір та кут обертання, швидкість затухання та появи частинок, налаштування внутрішнього освітлення (емісії), тощо.

Граф візуальних ефектів (VFX Graph) – це рішення, яке працює на GPU для імітації мільйонів частинок одночасно та створення масштабних візуальних ефектів. Граф візуальних ефектів також включає в себе візуальний редактор, який допоможе створити заздалегідь сконфігуровані візуальні ефекти. Тобто, VFX Graph слід використовувати для створення дійсно великих та реалістичних ефектів, таких як вибух будівлі, чи при моделюванні ситуацій інтеракцій космічних тіл. Крім того, конфігурації графа візуальних ефектів є портативними та можуть бути підготовлені до використання в сцені заздалегідь. Це означає, що не вдасться корегувати його роботу під час виконання програми, але в рамках опрацювання між сесіями навчання системи зміни є можливими. Також є можливість експортувати ефекти у шейдери, що дозволить використовувати їх біль загально [6].

Нижче на рисунку 1.6 наведено приклад створення вогню за допомогою інструментарію VFX Graph.

Рисунок 1.6 – Ефект вогню за допомогою VFX Graph

Та на рисунку 1.7 можна побачити створення вогню за допомогою стандартної системи частинок.

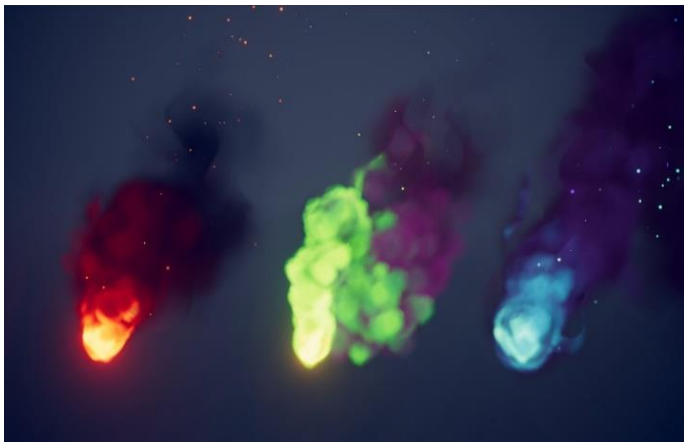

Рисунок 1.7 – Ефект вогню за допомогою стандартної системи частинок

На відміну від стандартної системи частинок, яка виконує всі операції на процесорі, VFX Graph автоматично є більш продуктивним у контексті рендерінгу зображення.

Отже, обидва рішення мають свої недоліки та переваги та можуть бути використані в залежності від поставленої задачі для генерації синтетичних даних. У контексті проблематики дослідження (для загального випадку) слід обрати саме використання стандартної системи створення частинок, бо таким чином буде здобута можливість конфігурувати параметри у реальному часі та не буде потрібно для кожної ситуації будувати вручну граф візуальних ефектів. Проте, для будування великих сцену у разі потреби покращення продуктивності слід використовувати саме граф візуальних ефектів.

У свою чергу система часток у UE4 обладає набагато просунітішим механізмом. Unreal Engine 4 має надійну та просту у використанні систему для створення ефектів частинок під назвою Cascade. Ця система дозволяє створювати модульні ефекти та легко контролювати поведінку частинок.

Розглянемо систему частинок вогню. В ній може бути задіяно декілька випромінювачів, таких як зображено на рисунку 1.8.

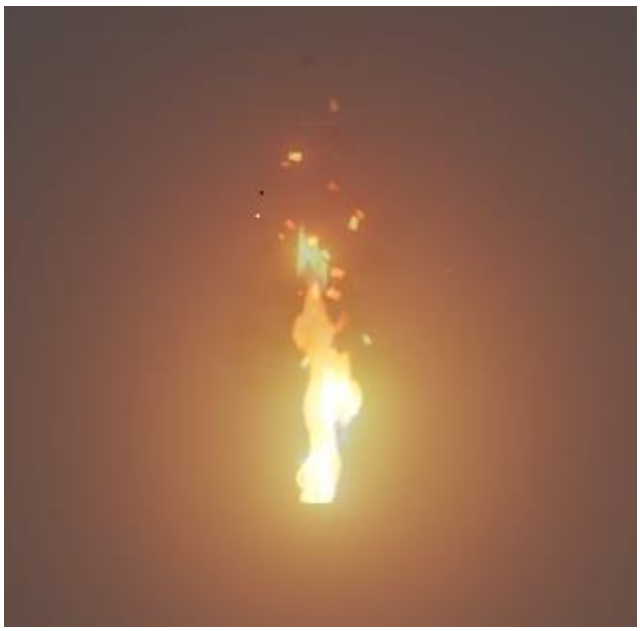

Рисунок 1.8 – Ефект вогню за допомогою декількох випромінювачів

Але коли камера знаходиться на великій відстані від джерела ефекту, то він виглядає, як зображено на рисунку 1.9.

У такому випадку є частинки, які занадто малі для відображення (менші за піксель), наприклад, вугілля. Однак ці частинки все одно будуть обчислені та оброблені.

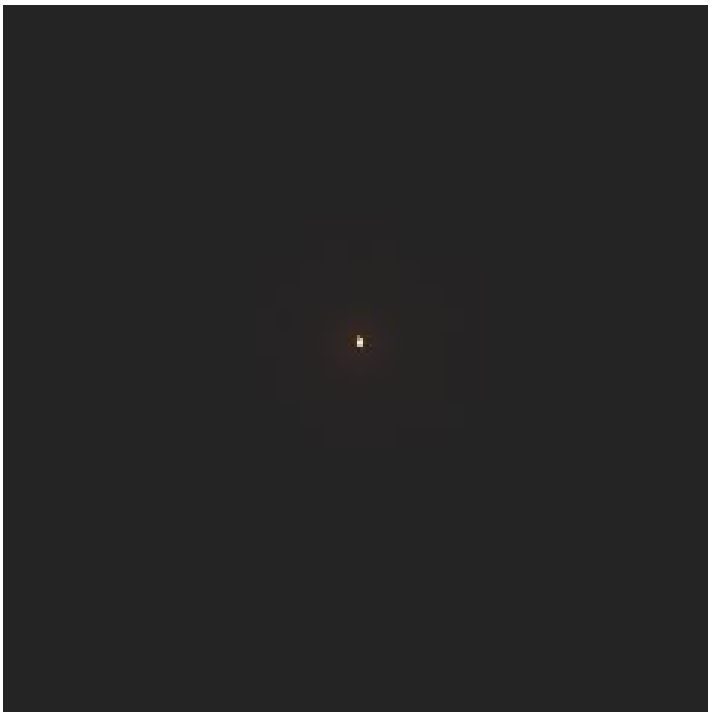

Рисунок 1.9 – Ефект вогню на великій відстані

Cascade дає результати рендерінгу в режимі реального часу та дає можливість редагування модульних частин ефектів, що дозволяє швидко та легко створювати навіть найскладніші ефекти.

Також, вона дозволяє динамічно встановлювати рівні деталізації (LODs). які забезпечують спосіб створення ефектів частинок, які максимально ефективно використовують простір екрану. Це місце, де слушно надходять рівні деталізації.

Рівні деталізації або LOD дозволяють спростити систему частинок на основі відстані, щоб випромінювачі та модулі в системі могли повністю їх відключити або відключити їх налаштування, як тільки камера буде занадто далеко, щоб ефективно побачити ефект.

Загальні причини та метрики низької продуктивності систем частинок:

- час оновлення (Tick time) – кількість часу, яке потрібно для ігрового потоку, щоб оновити всі системи;
- виклики малювання (Draw calls) – стан налаштування для підготовки до GPU;
- overdraw – кількість площі екрану, яку охоплюють частинки, і кількість інструкцій щодо цих частинок. Overdraw = Кількість шарів * кількість пікселів, на які вона впливає;
- розрахунок меж – кількість часу, необхідного для оновлення екрану;
- велика кількість полігонів при використанні геометрії;
- надання частинкам безпосередньої фізичної взаємодії за допомогою колайдерів;
- використання текстур або спрайтів великого розширення, наприклад, 2048x2048.

1.4 Механізми сегментації зображення

Одним з наступних аспектів у розробці практичного програмно забезпечення для створення синтетичних зображень слід зазначити також можливість дуже легко сегментувати зображення.

У теорії комп’ютерного зору сегментація означає процес поділу цифрового зображення на його складові регіони або об’єкти. Тобто, він розділяє зображення на окремі регіони, які мають на меті мати сильну залежність об’єктами чи особливостями. Сегментація зображення зазвичай має за собою ціль попередньою обробкою до розпізнавання образів зображення, вилученням та стисненням зображення, або загалом, це перший етап у будь–якій спробі аналізу або інтерпретації зображення автоматично. Успіх або невдача подальшої задачі з обробки зображень часто є прямим наслідком успіху або невдачі сегментації.

Сегментація зображень є завданням принципового значення в візуальному цифровому аналізі, і зазвичай це є перший крок у багатьох методах комп'ютерного зору. Власне це процес, що розділяє цифрове зображення на непересічні, не перекриваючі один одного регіони, кожен з яких зазвичай відповідає тому, що люди можуть легко відокремити та переглянути як індивідуальний об'єкт. На відміну від людського зору, де відбувається сегментація зображення на рівні мозку. Зазвичай без додаткових змін системи не мають засобів інтелектуального розпізнавання об'єктів, тому цифрова обробка вимагає, щоб ми дуже влучно ізолювали об'єкти розбиття зображення на регіони, по одному для кожного об'єкта. Основна мета сегментації – це розподіл зображення на набір непересічних областей, які візуально відрізняються одна від іншої, але внутрішньо однорідні та змістовні, мають спільні деталі та властивості. Наприклад, рівень сірого, колір, текстура, глибина, рух, тощо. Такий спрощений вигляд набагато простіше надалі аналізувати та використовувати.

Через те що ми маємо безпосередній доступ до процесу рендерінгу – маємо можливість отримати інформацію про кожен ігровий об'єкт – його позицію, компоненти застосовані до нього, чи знаходиться він у полі зору камери тощо. Запровадження сегментації можливо безпосередньо за допомогою техніки підміни матеріалів чи шейдерів. Матеріали в 3D–движках, як було згадано раніше – це набір карт, який інтерпретується фізичним модулем движка та використовує певні його властивості для коректного відображення, програму яка певним чином інтерпретує текстури називають шейдерами [7]. Динамічна зміна цих матеріалів допоможе створити маску сегментації. Мета – замінити кожен матеріал на картах лише кольором (не беручи до уваги жодних фізичних властивостей під час створення зображення), де всі елементи одного класу матимуть той самий колір. Розглянемо етапи цього процесу з точки зору практичного застосування у Unity:

- обудувати альтернативні матеріали (один колір у класі);
- позначити Unlit shading для матеріалу (не використовувати підрахунки освітлення);

- призначити освітлення рівномірного кольору (тобто без градієнту);
- матеріали з прозорих частками зробити дещо складніше.маємо зберегти прозорість для кожного пікселя, який мав би бути прозорим у звичайному вигляді сцену;
- встановити імена тегів ігровіх об’єктів у сцені (розбиття на класи);
- створити таблицю даних, у якій перераховані всі теги дійових осіб, індекс матеріалу, матеріал за замовчуванням та альтернативний матеріал;
- створити події, які вибирають усі ігрові об’єкти за назвою групи та змінювати матеріал на альтернативний.

Приклад зображення обраної сцени до процесу сегментації можна побачити на рисунку 1.10.

Рисунок 1.10 – Несегментоване зображення локації

Отримавши сцену зі створеною локацією – надаємо теги, класи до кожного унікального елементу та поставимо у відповідність альтернативний матеріал з

кольором та змінемо його, після чого створене зображення можна бути зберегти до файлової системи.

На результуючому зображенні маємо можливість побачити точне розподілення різних об’єктів за класами та надання їм альтернативного матеріалу з унікальним кольором в рамках сцени. Це можна побачити на рисунку 1.11.

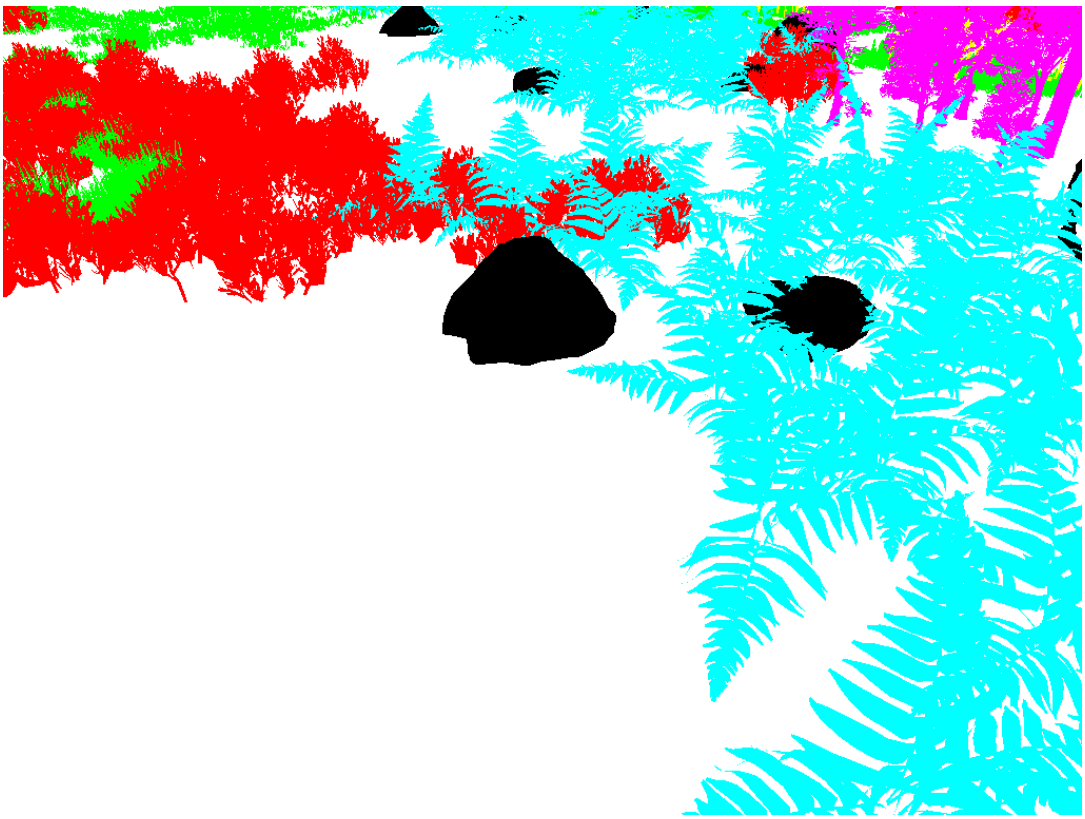

Рисунок 1.11 – Сегментоване зображення локації

Великою перевагою використання зображень, створених штучно, як навчальних даних, крім заміни ручного фотографування та / або збору, є простота створення обмежувальних сегментів. У Unity ці обмежувальні поля можна сформувати, запроектувавши всі вершини 3D–моделі в 2D–простір екрану, а потім витягнувши мінімальне та максимальне положення X та Y всіх вершин.

Семантична анотація необхідна разом із зображеннями RGB для створення наборів даних для систем по детекції об’єктів на основі машинного навчання. Особливо для великих (синтетичних) наборів даних, навчальні дані повинні бути

створені автоматично та швидко. Стратегії маркування або анотації можуть бути фактором витрат, якщо виправлення або видалення небажаних даних. Також можна зазначити такі додаткові техніки обробки результуючого зображення.

По перше, можливість категорізувати об’єкти. При створенні сцени ми маємо пряму можливість додати теги до кожного об’єкту, тобто визначити його категорію. Надалі для кожної категорії можемо анотувати унікальний колір.

По–друге, можливість легко визначити оптичний потік. Оптичний потік – це прогноз для кожного піксель, головна ідея якого полягає в тому, що він передбачає динаміку зміну яскравості. Тобто намагається оцінити, як яскравість пікселів рухається по екрану з плином часу. Через той факт, що оптичний потік можна застосувати тільки у динаміці – його використання релевантно виключно для синтетичних відео. Це корисно для забезпечення згладжування в генеративних змагальних мережах (GAN), щоб згенерований результат міг бути тимчасово зв’язаним. Це буде важко зробити виключно з використанням GAN, які не закодовують компонент плина часу.

По–третє, це визначення глибини для об’єктів зображення. У цьому випадку пікселі змінюються в залежності до відстані від камери до об’єкту сцени. Деякі приклади застосування оцінки глибини включають в себе вирівнювання розмитих частин зображення, покращення візуалізацій 3D–сцен, автоматичне перетворення з 2D на 3D у фільмі та підрахунки тіней. А також відображення нормалей – у цьому випадку поверхні забарвлюються відповідно до їх орієнтації стосовно камери. Приклад застосування цих технік в рамках Unity можна побачити на рисунку 1.12.

Звичайно, ці механізми за замовчування доступні, бо під час виконання програми наявний доступ до повного стану системи у будь–який час, та не потребується жодних додаткових алгоритмів по перетворенню зображення. Однак, такий ефект досягається за допомогою ImageSynthesis шейдеру, який входить до пакету інструментів UnityML. Це допомагає виконувати ці операції узагальнено для будь–якої конфігурації сцени. Наприклад, для пошуку оптичного потоку у відео використовується OpticalFlow.shader шейдер, котрий використовує спеціалізовану кольорову палітру, де похідний чи початковий колір є чорним, а не

білим, як це є у інших кольорових палітрах. Також використовується саме HSV кодування кольору.

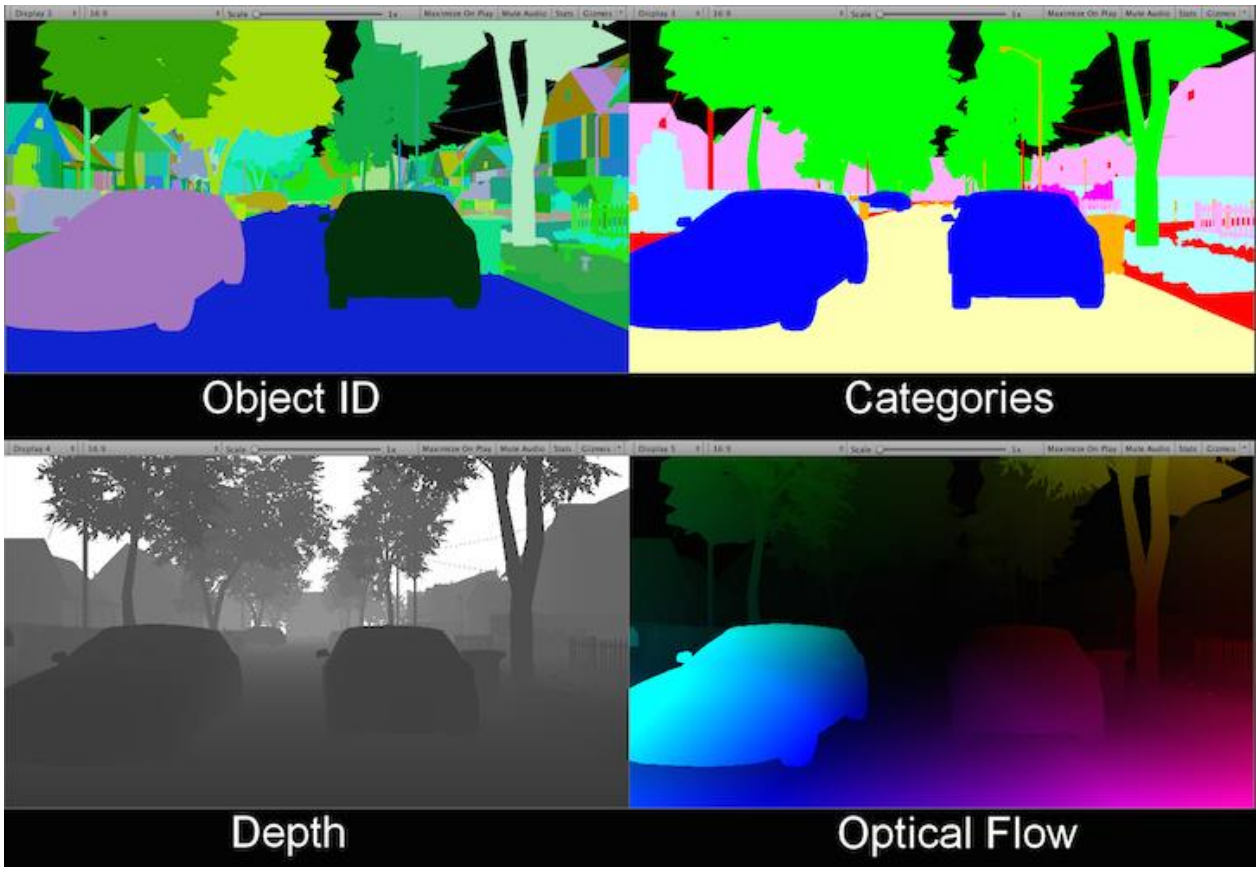

Рисунок 1.12 – Використання категорізації, визначення глибини та оптичного потоку

Резюмуючи усі попередні пункти дослідження, до переваг генерування та використання синтетичних зображень у рамках навчання (в загальному сенсі) системи можемо віднести:

- можливість зміни кута, положення, поля зору камери;
- можливість зміни обертання, масштабу, рівня деформації 3D–об’єкта тощо;
- можливість динамічною зміни кольору та текстури;
- зміна умов освітлення в приміщенні та на вулиці;
- контролювання дня та ночі;

- контроль рівня шуму;
- високоточна анотація;
- можливість генерації поглибленої карти та хмари точок з імітацією всіх параметрів 3D–зондування;
- ненормальний обсяг даних;
- контроль за кількістю об’єктів або наявністю будь–яких компонентів та властивостей об’єкта;
- контроль за перешкодами та видимістю додаткових об’єктів.

Отже, згрупувавши теоретичні відомості та гіпотези щодо впливу різних параметрів налаштування сцени на створенні синтетичних зображень перейдемо безпосередньо до підтвердження їх емпірично.

2 ПРОЕКТУВАННЯ ТА ОПИС РОЗРОБЛЕНОЇ ПРОГРАМНОЇ СИСТЕМИ

2.1 Генерація зображень

Зваживши всі перелічені аспекти було отримано узагальнені вимоги до створення прототипу для перевірки висунутих гіпотез. Також, при порівнянні двох доступних рішень, був обраний саме Unity, за тієї причини, що він надає гнучкі можливості в контексті балансування між оптимізацією та якістю створених зображень.

Умови освітлення – це один із факторів, що сильно відрізняється від реального світу під час використання ігрових движків, тому для створення набору даних із різними умовами освітлення було використано наступні налаштування.

Сцени облаштовані небом (скайбоксом), включаючи світло півкулі, світло у точці та світло з напрямком. Світло півкулі є повністю статичним і імітує світло, відбите від землі, а також сонячне світло з неба. Далі, джерело світла у точці, світло, що світить у всіх напрямках, розміщується над видимим об'ємом виду. Таке джерело світла вмикається т вимикається для різних зображень з визначеною певною ймовірністю.

Світло з напрямком обертається випадковим чином навколо осі Y, щоб гарантувати, що об’єкти освітлюються не в постійному напрямку. Комбінація таких типів освітлення дозволить наближити сцени до більш реалістичних та динамічних, а також збільшити кількість доступних для синтетичного створення зображень. Додатковим аспектом у контексті освітлення є використання карт внутрішнього освітлення. Вони є більш ефективними з точку зору швидкості виконання, але не досить реалістичними та масштабними, щоб освітлювати усю сцену. Її можна використати, наприклад, для світла з фар чи для підкреслення певних об’єктів [8].

Доцільно також розглянути формати можливі формати файлів для використання та їх можливі переваги та недоліки.

З точки зору моделей існує два найпопулярніші формати. Перший, це OBJ набагато простіший формат, який зберігає лише інформацію про геометрію (вертекси, нормалі тощо) та дані UV розгортки.

Другий, FBX – це набагато більш просунутий формат, який може вмістити більше даних – моделі, дані UV розгортки, інформацію про анімацію, багаторазове використання анімації в одному файлі, NURBS криві та навіть вбудовувані файли текстур. Тому, якщо не використовувати анімацію рухів об’єктів, то, з точки зору оптимізації процесу, набагато ефективніше використовувати саме OBJ формат.

Формати зображень текстур – це питання балансування між розміром файла та ступенем втрат якості зображення.

У TGA найважливіше – це здатність зберігати «прозорість» – тобто те, наскільки «багато» кольору піксель показує, як непрозорий, та наскільки вона дозволяє пропускати все, що знаходиться «позаду». Цей формат займає навіть більше місця, ніж BMP у звичайних ситуаціях, але може підтримувати стиснення (але зазвичай це не так).

PNG схожий на TGA тим, що він є форматом без втрат якості. Як і TGA, він має підтримку прозорості. Ключова відмінність полягає в тому, що це більш сучасний формат і він завжди стискається. Це означає, що за замовчуванням він буде займати значно менше місця, ніж BMP або TGA.

JPEG – це формат зберігання зображень з втратами якості. Використання зображення у форматі Він завжди матиме загублені дані. Навіть, якщо таке зображення буде змінено лише частково, то після повтороного збереження, воно втратить ще більше даних та якості.

Робимо висновок, що PNG – найкращий вибір, бо немає втрат у якості зображення та займає менше місця ніж TGA за рахунок стиснення на етапі створення.

Об’єкти оточення та автомобілі використовують основну фізично–коректну послідовність рендерінгу з використанням карт нормалей, висоти, металевості та жорсткості, для того щоб надати поточним сценам реалістичний вигляд. Як і було описано у попередній частині дослідження ефекти вогню та диму будуть

побудовані на класичній системі часток, з використанням спрайтів, щоб мати повний контроль за динамікою відтворення ефекту. Зваживши ці вимоги можемо перейти до етапу проектування основних модулів програмного продукту. Структуру основних компонентів генератора зображень можна побачити на рисунку 2.1.

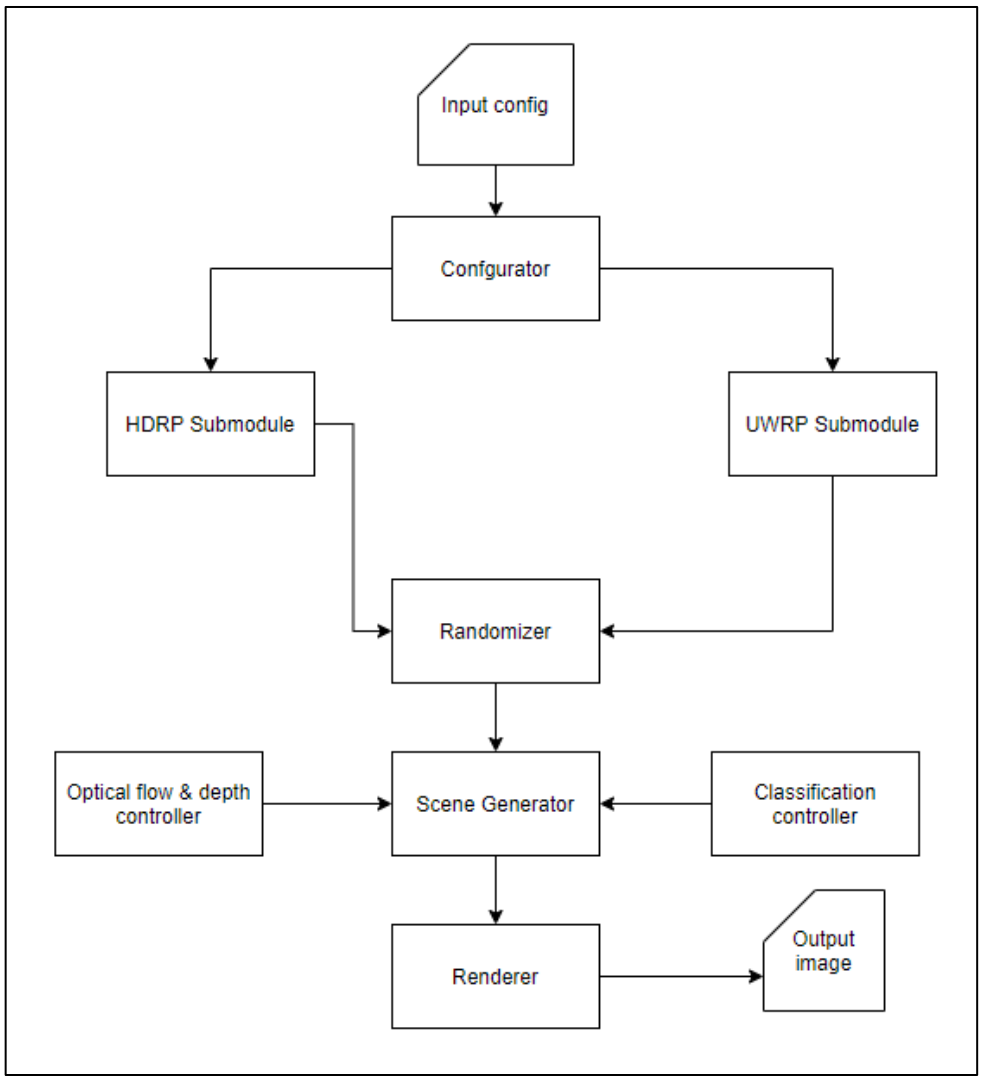

Рисунок 2.1 – Структура модулів програмної системи генератора

Компонент Configurator отримує на вхід файл за яким він визначає використовувати послідовність за допомогою HDRP чи UWRP.

Компонент HDRP Submodule відповідає за генерацію сцен з використанням пайплайну рендерінгу високої якості.

Компонент UWRP Submodule відповідає за генерацію сцен з використанням пайплайну рендерінгу оптимізованого під мобільні пристрої.

Компонент Randomizer на вхід отримує дані від одного з двох підмодулей та визначає границі випадковості з конфігураційного файлу та створює псевдовипадкові значення та передає їх до компоненту Scene Generator.

Scene Generator розташовує усі потрібні об’єкти та налаштування у сцені та передає команду до компоненту рендерінгу.

Renderer безпосередньо створює зображення у заданому розширенні.

Якщо модулі Optical Flow And Depth Controller та Classification controller активовані вони також направляють команди до компоненту рендеринга зображення, аби створити категоризовану версію зображення чи оптичний потік відео. Результуючий застосунок може контролюватися за допомогою передавання шляху до конфігураційного файла для запуска процесу генерації. З використанням цього файлу можна налаштовувати усі аспекти, які були досліджені та можуть впливати на результати швидкості рендерінгу та навчання (наприклад – налаштування камери, світла, використовувати сегментацію чи ні, тощо).

Додатково є можливість використати інструменти ImageSynthesis та згенерувати також зображення глибини, нормалей та категорій. Через той факт що практична реалізація не підтримує створення синтетичних відео–оптичний потік побудований не буде. Проте була створена реалізація шейдеру для цієї задачі за допомогою інструментарію движка для створення фізичних матеріалів. Код , який це демонструє наведений нижче:

```
float3 MotionVectorsToOpticalFlow(float2 motion) {
    float angle = atan2(-motion.y, -motion.x);
    float hue = angle / (UNITY_PI * 2.0) + 0.5; // convert motion angle
to Hue
    float value = length(motion) * _Sensitivity; // convert motion
strength to Value
    return HSVtoRGB(float3(hue, 1, value)); // HSV -> RGB
}

fixed4 frag(v2f i): SV_Target {
    float2 motion = tex2D(_CameraMotionVectorsTexture, i.uv).rg;
    float3 rgb = MotionVectorsToOpticalFlow(motion);
    return float4(rgb, 1);
}
```

Ця функціоналість шейдеру дозволяє трансформувати вектори зміни положення об’єктів в зображення оптичного потоку.

2.2 Генерація відео

Розглянемо також можливості Unity в рамках створення синтетичних відео послідовностей та спроектуємо взаємодію з точки зору нашої системи. У розлянутому контексті центральним елементом фреймворку для запису синтетичних відео є компонент редактор сценаріїв – Scenario Editor. Цей редактор відповідає за налаштування сцени з потрібними параметрами, а також запуск та зупинку самого процесу запису. Вхід користувача або файл із потрібними параметрами надається безпосередньо редактору сценарія. Звідти налаштування розподіляються у відповідності до потрібних ігрових об’єктів, таких як світло.

Коли запис відео розпочинається, йде у хід екземпляр класу Custom Recorder, який відповідає за всі графічні виходи, а саме за рендерінг та збереження синтетичних зображень, а також відповідної їх анотації. Додатково до результатів користувач також ініціалізує об’єкт запису виходів, який керує виводом. На цьому етапі встановлюються параметри сцени, що надаються в редакторі, а також перелік критичних подій, записаних спеціальною сутністю, зберігаються у файлі JSON.

Параметри сценарію надаються двома способами: налаштування навколишнього середовища, такі як час доби, швидкість вітру, а також установки після обробки, такі як шум, розмиття та роздуття, додаються у якості параметрів середовища або завантажуються. через файл JSON. Інші налаштування, наприклад, точки для навігації персонажа і точки створення об’єкту камери визначаються в редакторі Unity. Додавання нових камер можливо також через редактор.

Оскільки відповідність реальних представлень камер у Unity є одним із пріоритетів у побудуванні цього фреймворку, вигляд камери може бути відрегульований у вікні перегляду Unity, а потім вони будуть перевикористані у

якості префабів (конфігурацій об’єктів збережених заздалегідь). Об’єкт камери, який називається Camera Controller в рамках цього фреймворку, складається з трьох ігрових об’єктів.

Перший – це батьківський об’єкт камер, який обробляє всі перетворення для камери. У цього батьківського ігрового об’єкта є два дочірні об’єкти камери Unity, які відповідають за запис сцени. Об’єкт Main Camera відображає сцену з налаштуваннями візуалізації з ігрового движка. Всі ефекти пост–обробки додаються саме до цього об’єкта.

Іншим дочірнім об’єктом контролера камери є так звана анотаційна камера, яка відповідає за надання інформації мітки для всіх об’єктів поточної сцени. Для цього всі шейдери в сцені замінюються чорним, не освітленим матеріалом, а всі об’єкти із властивістю кольору мітки виводяться у цьому єдиному кольорі мітки. Збереження кадрів сцени у файли відбувається під час рендерінгу. Цей процес виконується для кожної камери у всіх контролерах камери сцени. Коли в сцені породжується новий ігровий об’єкт, йому призначається унікальний колір мітки та відзначається у вихідному мета файлі. Обрані кольори в будь–якій даній сцені вибрані таким чином, щоб вони мали якомога більшу відстань у відтінку, що дозволяє обробляти мітки навіть у стиснутих форматах відео.

Під час розглядання процесу створення синтетичних відео застосовуються ті ж самі техніки та спостереження, що і для зображень. Однак, ще одним додатковим фактором є наявність потреби додаткового процесорного часу на зберігання кожного поточного кадру та запис у файл. Це достатньо знижує ефективність та кількість можливих відмальованих кадрів у одиницю часу. Операції користувача у Unity працює у одному потоці за замовчуванням, хочу є можливість використати вбудовану системну систему по розподіленню задач, щоб запис та рендерінг були незалежними. Розглянемо також інші супутні компоненти [9]:

– CarWaypoints відповідає за розташування об’єктів автомобіля у сцені;
– SunController відповідає за налаштування глобального освітлення та небесної коробки сцени, зміну дня та ночі;

– LightController контролює інтенсивність освітлення та кількість локальних точок освітлення у сцені;
– OutputRecord записує відео безпосередньо на жорсткий диск;
– TriggerController приймає команди до виконання та конфігураційний файл.

За перелічиними компонентами, котрі будуть використовуватися була побудована узагальнена структура запису синтетичного відео, яку можна побачити на рисунку 2.2.

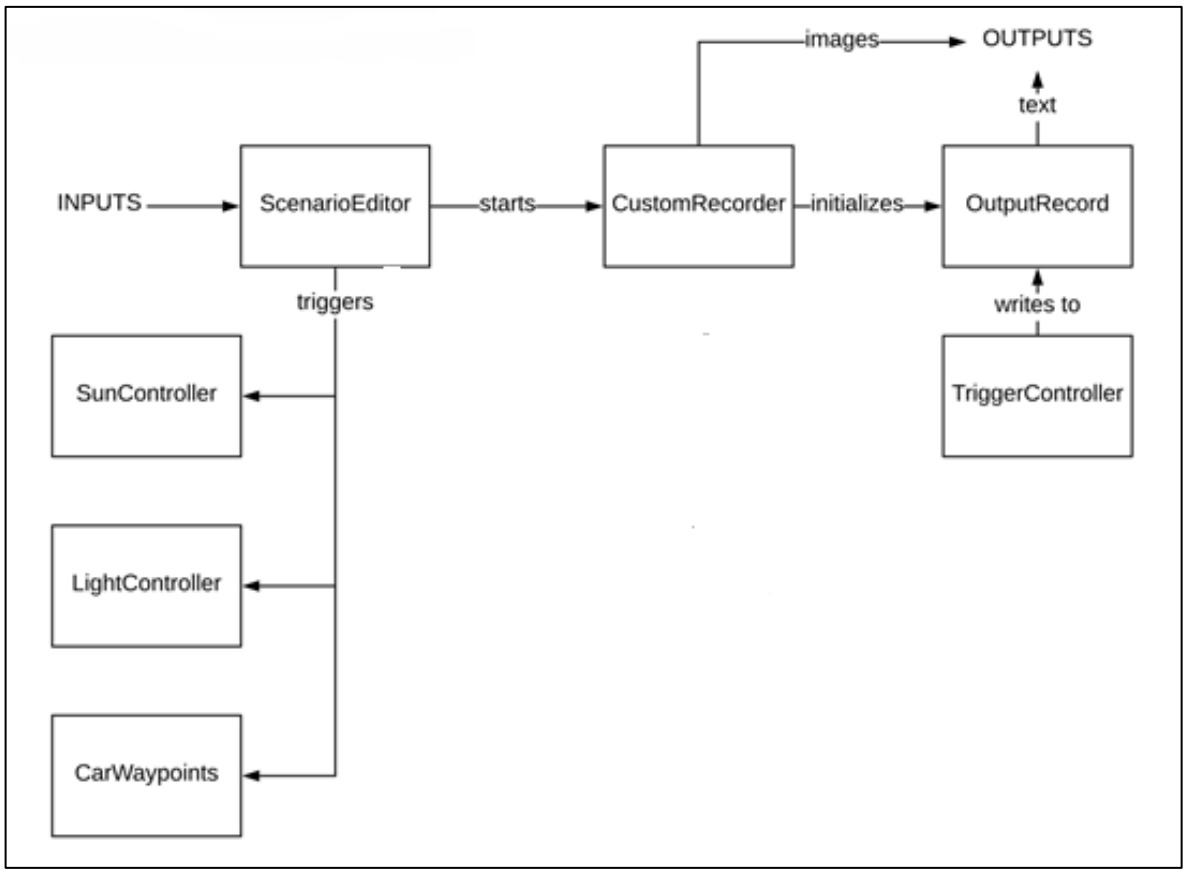

Рисунок 2.2 – Програмна структура запису синтетичного відео

За переліченими критеріями, які можна легко сконфігурувати (камера, світло тощо), був створений програмний продукт для послідовної генерації зображень для навчання моделі передбачення пожежі. Зміна цих налаштувань доступна за рахунок використання параметрів запуску командної строки. Нижче наведений приклад коду для цього випадку:

```
public class BuildScript
{
    private static readonly string configPath;
    static BuildScript()
    {
configPath =Environment.GetEnvironmentVariable("CONFIGURATION");
        if (string.IsNullOrEmpty(configPath))
            configPath = "/sceneConfig.scene";
    }
}
```

Таким чином можливио викликати створений застосунок за допомогою консольного терміналу операційної системи виконавши команду:

```
./imgen.x86_64 configuration="configPath"
```

Це надає можливість використовувати цю утиліту при запусках на віддалених серверах у хмарових сервісах через інші застосунки. Таким чином був спроектований та створений додаток, який можна використовувати для створення синтетичних даних, безпосередньо розглянемо його у розрізі використання його в рамках цього дослідження.

3 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

Розроблений програмний продукт був зконфігурований для використання у контексті класифікації автомобілем та пожеж в автомобілях. Були створені певні моделі та текстури для них, налаштовано світло та власне сцена. Одне з отриманих зображень можна спостерігати на рисунку 3.1.

Рисунок 3.1 – Згенерована сцена

В сцені були використані системи частинок, зміна куту нахила камери, наближення камери, декілька джерел світла, зміна інтенсивності світла, динамічна зміна текстур тощо. Для створення файлів зображення було використано стандартні методи захоплення EngineScreenCapture [10].

У створеному демо є можливість динамічно змінювати сцену, додаткові моделі навколишнього середовища, а також об’єкт, який модель буде використовувати для класифікації.

Також, за низькою фактів був обраний хід роботи рендеру – UWRP. Дана система дозволяє згенерувати таку кількість створених зображень (у контексті виявлення автомобіля та пожежі у автомобілі):

– 135 зображень з розширенням 800x600 за одну хвилину;

– 71 зображень з розширенням 1280*920 за одну хвилину.

Але, для кожного зображення та сцени, результати будуть сильно відрізнятись. Саме тому, для кожного випадку, треба робити індивідуальний профайлінг результатів.

Розглянемо приклад згенерованого зображення за низкою динамічних факторів, який можна побачити на рисунку 3.2.

Рисунок 3.2 – Сцена з меншою кількістю джерел світла

Розроблена система змінює фактори псевдо–випадковим способом за допомогою генерації випадкової величини з урахуванням ваги. Це є значення, яке ми обираємо із низки однакових сутностей, які мають різні вірогідності.

Можна побачити, що основна деталізація не була загублена, але кольорова палітра значно змінилася, тому можна зробити досить очевидний висновок, що генерація зображень з меншою кількістю світла буде ефективніша для класифікації зображень уночі чи ввечері.

Приклад генерації зображення зі зміненим оточенням навколо основного об’єкту можна побачити на рисунку 3.3, який наведений нижче.

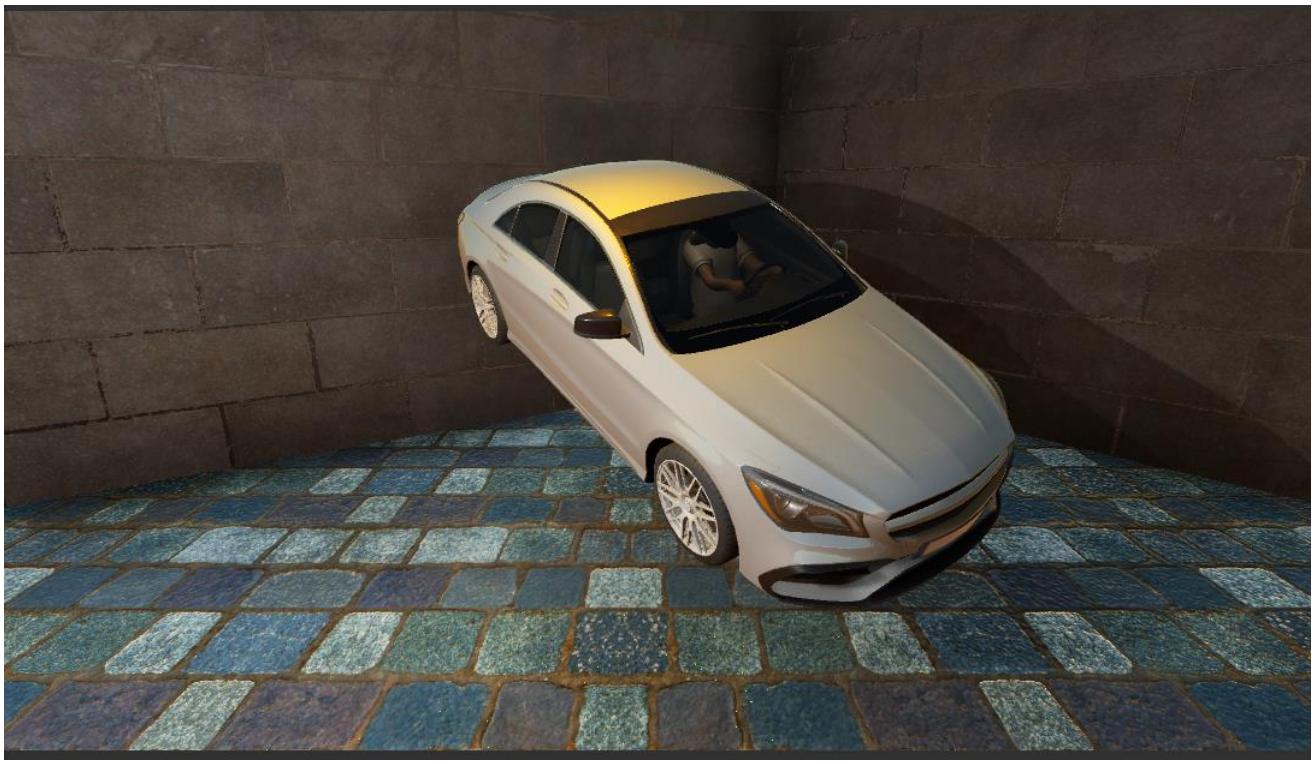

Рисунок 3.3 – Сцена зі зміненим оточенням навколо основного об’єкту

Щодо використання різноманітного розташування UV–сетів для об’єктів у сцені, то істотних змін при використанні, наприклад, одного UV–сету на всю сцену, чи окремих для кожної деталі, спостережено не було. Цей момент відіграє ключову роль саме при створенні потрібних нам об’єктів для синтетичної генерації, що не входить у межі даного дослідження.

Для перевірки можливостей ефективно сегментувати зображення – використаємо процедурно згенеровані локація за допомогою тайлів (елементів оточення однакового розміру котрі можна комбінувати у будь–якому порядку).

Слід використовувати псевдовипадкову генерацію, використовуючи тайли. Повністю випадкове створення набагато складніше, якщо задачею є створити зображення, які мають великий ступінь реалістичності [11].

Розглянувши безпосередньо механізм сегментації – розглянемо також техніку створення такого зображення. Для цього будемо зміщувати камеру за спіраллю (одночасна зміна позиції та куту нахилу камери) та послідовно робити скріншоти сцени у звичайному стані та з альтернативними матеріалами. Однак, слід уважно слідкувати, щоб усі зміни до матеріалів закінчилися до кінця, для того щоб запобігти артефактам на результуючому зображенні. Приклад цього експерименту – отримані файли на жорсткому диску можна побачити на рисунку 3.4.

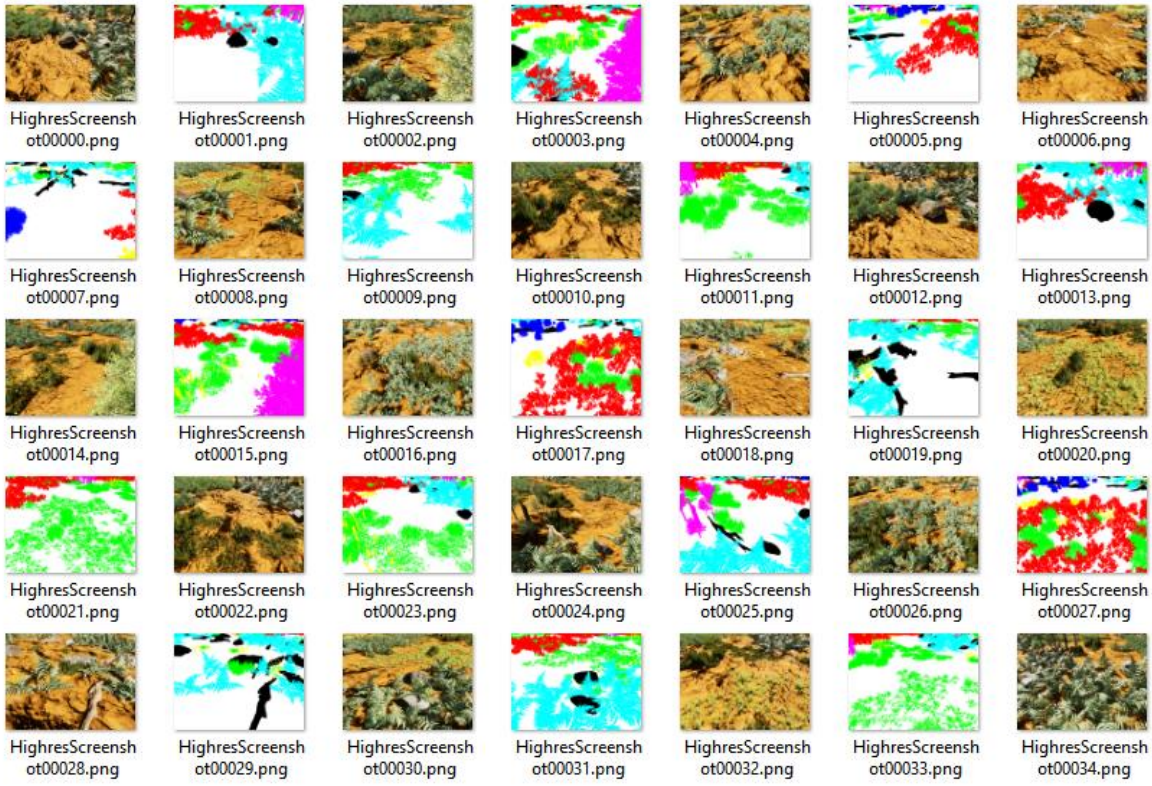

Рисунок 3.4 – Результуючі синтетичні зображення та альтернативні, сегментовані

Надалі ці дані можна легко використати під час тренувань систем машинного навчання для класифікації зображень інтегруючи маскування.

У якості завершального етапу розгляду практичної апробації даного дослідження слід розгляну конфігурацію та створення мета даних, якими супроводжується синтетичні відео та зображення [12]. З точки зору формату даних – уся інформація може бути отримана з поточного стану системи. Наприклад, таким чином як показано на коду нижче:

```
--- !u!1 &1534488194 stripped
GameObject:
  m_CorrespondingSourceObject:   {fileID:   1648972838963176,   guid:
35397e79fbf6a3845b4b5400fac263f9,
    type: 3}
  m_PrefabInstance: {fileID: 1534488192}
  m_PrefabAsset: {fileID: 0}
--- !u!1 &1534488196 stripped
GameObject:
  m_CorrespondingSourceObject:   {fileID:   1840166929206288,   guid:
35397e79fbf6a3845b4b5400fac263f9,
    type: 3}
  m_PrefabInstance: {fileID: 1534488192}
  m_PrefabAsset: {fileID: 0}
--- !u!1 &1534488198 stripped
GameObject:
  m_CorrespondingSourceObject:   {fileID:   1573027431656734,   guid:
35397e79fbf6a3845b4b5400fac263f9,
    type: 3}
  m_PrefabInstance: {fileID: 1534488192}
  m_PrefabAsset: {fileID: 0}
--- !u!114 &1570741040 stripped
MonoBehaviour:
  m_CorrespondingSourceObject:   {fileID:   114499213394813874,   guid:
931c8bb3cff96bf4b98a4516b634d539,
    type: 3}
  m_PrefabInstance: {fileID: 1039350533}
  m_PrefabAsset: {fileID: 0}
  m_GameObject: {fileID: 0}
  m_Enabled: 1
  m_EditorHideFlags: 0
  m_Script:           {fileID:           -765806418,           guid:
f70555f144d8491a825f0804e09c671c, type: 3}
  m_Name:
  m_EditorClassIdentifier:
--- !u!224 &1592349521 stripped
```

Для кожного об’єкту сцени є можливість отримати застосовані до нього компоненти, їх ієрархічну структуру, а також посилання на інші компоненти системи чи налаштування. Отримати такі дані можна у любий момент часу під час рендерінгу, тому слід генерувати зображення чи відео завжди разом з подібним мета–файлом.

ВИСНОВКИ

Підбиваючи підсумки та аналізуючи результати та хід проведеного дослідження слід зазначити, що були систематизовані та класифіковані метрики та інструменти генерації візуальних синтетичних даних. З точки зору ключових показників, були сформовані залежності між різноманітними техніками рендерінгу, налаштуваннями сцени, рівнем деталізації, використанням систем часток для оптимального використання разом з системами машинного навчання.

У рамках дослідження способів оптимізації процесу були знайдені ефективні та практичні закономірності між використанням певних форматів файлу, режимів роботи рендеру ігрового движку та ефектів пост–обробки.

Було проведено порівняння двох відкритих ігрових движків Unity та UE4 в аспекті інструментів створення фізично–реалістичних симуляцій та їх переваги над іншими механізмами та способами створення синтетичних даних, наприклад, доповнення зображень з реального світу новими об’єктами.

Крім того, була створена програмна реалізація, яка дозволила експериментально підтвердити вплив усіх перелічених параметрів на швидкість створення даних та удосконалення якості навчання моделі. За допомогою створеної системи була апробована практична значимість цього дослідження, а саме можливість створити такий тип зображень чи відео, який фізично нерентабельний з реальними даними.

Якщо розглядати можливий розвиток у напрямку взаємодії створеного генератору синтетичних даних з іншими системами машинного навчання, то наступним кроком у розвитку цього дослідження може стати створення методів уніфікованого налаштувань під будь–які вхідні умови з боку користувача, а також впровадження автоматичного, «розумного» удосконалення конфігурацій у реальному часі (використовуючи статистичні результати прогнозів).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Medium. URL: https://medium.com (дата звернення: 01.05.2020).

2. Unity Documentation. URL: https://docs.unity3d.com/Manual/ (дата звернення: 01.05.2020).

3. Apollo Auto Data Set. URL: http://apollo.auto/ (дата звернення: 01.05.2020).

4. Unreal Engine 4 Documentaion. URL: https://docs.unrealengine.com/en–US (дата звернення: 01.05.2020).

5. A. Vechur, A. Chayka, Y. Chemodakov. The component method of scene analysis and object recognition: стаття. США, IEEE, 2003 - 5 с.

6. Richard Hoptroff. Practical Synthetic Data Generation: книга. США, Каліфорнія: O'Reilly Media, Inc., 2019 – 543 с.

7. Matt Pharr, Wenzel Jacob, Greg Humphreys. Physically Based Rendering, Third Edition: From Theory to Implementation: книга. Нідералнди, Амстердам: Elsevier Inc., 2017 – 1266 с.

8. І. А. Ревенчук. Математичні моделі геометричних перетворювань при візуалізації 3D об'єктів: стаття. Україна: Вісник НТУ «ХПІ», 2013 – 4 с.

9. Tomas Akenine Möller, Eric Haines, Naty Hoffman. Real–Time Rendering, Fourth Edition: книга. США, Флоріда: CRC Press, 2018 – 1199 с.

10. Aurélien Géron. Hands–on Machine Learning with Scikit–Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems: книга. США, Каліфорнія: O'Reilly Media, Inc., 2019 – 932 с.

11. С. Ю. Шабанов, Ю. С. Новіков. Представление знаний сложного структурируемого объекта в задачах диагностирования с использованием моделей: стаття. Україна: Вісник НТУ «ХПІ», 2016 – 4 с.

12. Alex Okita. Learning Programming with Unity: книга. США, Флоріда: CRC Press, 2014 – 732 с.